



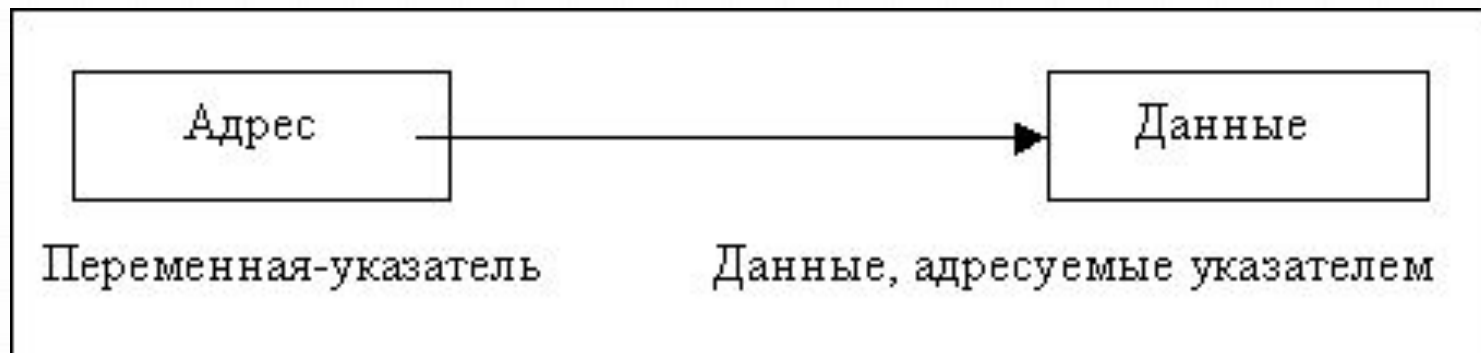
& *

Указатели позволяют

- 0 обрабатывать многомерные и одномерные массивы, строки, символы, структуры и массивы структур.
- 0 динамически создавать новые переменные в процессе выполнения программы.
- 0 обрабатывать связанные структуры: стеки, очереди, списки, деревья, сети.
- 0 передавать функциям адреса фактических параметров.
- 0 передавать функциям адреса функций в качестве параметров.

Определение

0 Указатель-это переменная или константа, которая содержит значение адреса другой переменной.



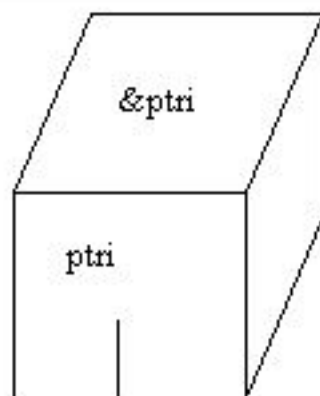
Объявление указателей и основные операции над ними

- 0 тип [модификатор] *<имя-указателя>
- 0 тип-имя типа переменной, адрес которой будет содержать переменная- указатель.(например integer, char, long)
- 0 имя-указателя –идентификатор переменной типа указатель.(имя собственное)
- 0 *- определяет переменную типа указатель.

- 0 Значение переменной-указателя-это адрес некоторой величины, целое без знака.
- 0 Указатель содержит адрес первого байта переменной определённого типа.
- 0 Тип адресуемой переменной, на которую ссылается указатель, определяет объём оперативной памяти, выделяемой переменной, связанной с указателем.

- 0 указатель содержит адрес нулевого байта этой переменной
- 0 тип адресуемой переменной определяет, сколько байтов, начиная с нулевого (адреса, определённого указателем) занимает это значение

Примеры объявлений указателей

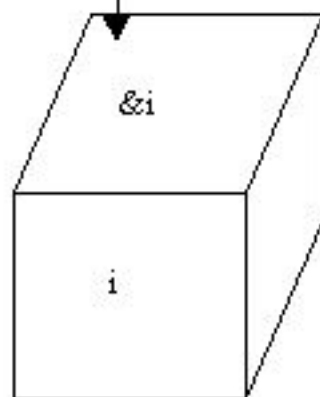


`&ptri`

-адрес указателя.

`ptri=&i;`

-значение указателя, адрес `i`.



`&i`

-адрес переменной `i`.

`i=*ptri;`

-значение переменной.

& И *

- 0 &-получение адреса переменной.
- 0 *-извлечение значения, расположенного по этому адресу.

&-имя переменной

- 0 получение адреса, **определяет адрес** размещения значения переменной определённого типа. Операндом операции **&** должно быть имя переменной того же типа, для которой определён и указатель левой части оператора присваивания, получающий значение этого адреса.

*-ИМЯ указателя

0 получение значения определённого типа по заданному адресу. Определяет содержимое, находящееся по адресу, который содержится в указателе-переменной или указателе-константе. Иначе: **косвенная адресация.**

Косвенная адресация

0 помощью операции * осуществляет доступ к значению по указателю, то есть извлечение значения, расположенного по адресу-содержимому указателя. Операнд * (т.е имя после) должно быть типа указатель (ранее объявлено).

Инициализация указателя

0 имя указателя_переменной=&имя_переменной

0 int *ptri,i;

0 //объявление указателя и переменной типа int

0 ptri=&i;

0 //ptri получает значение адреса 'i'

0 оператор присваивания, использующий имя указателя и * операцию косвенной адресации:

0 **Имя_переменной=*имя_указателя**

0 Имя указателя –это переменная или константа, которая содержит адрес размещаемого значения, требуемого для переменной левой части оператора присваивания

0 **i=*ptri;**

0 // 'i' получает значение, расположенное по адресу

0 // содержащемся в указателе 'ptri'

Взаимосвязь указателя, адреса и значения переменной

```
int *pi           //указатель-переменная на данные типа integer
char *pc          //указатель-переменная на данные типа char
float *pf         //указатель-переменная на данные типа float
int m1[5]         //объявление массива m1 типа int на 5 значений
                 //m1-указатель-константа
int *m2[10]       //m2-имя массива на 10 значений типа указатель на
                 //значения типа int
```

Указатели можно ИСПОЛЬЗОВАТЬ

- 0 ***ptri**-значение переменной, находящейся по адресу, содержащемуся в указателе ptri
- 0 **ptri**-значение адреса переменной
- 0 **&ptri**-адрес местоположения самого указателя

```
0 int i=123, j, *ptri;  
0 //объявление переменных и указателя  
0 ptri=&i;  
0 //инициализация указателя(присвоение адреса i)  
0 j=*ptri+1;  
0 //переменной i (*ptri) присваивается значение  
0 //переменной i и к её содержимому прибавляется  
    единичка.
```


Многоуровневая адресация

0 **int i=123;**

0 //где i-имя переменной

0 **int *pi=&i;**

0 //pi –указатель на переменную

0 **int **ppi=π**

0 //ppi-указатель на ‘указатель на переменную’

0 **int ***pppi=&ppi;**

0 //pppi-указатель на ‘указатель на ‘указатель на переменную’.

Правила

- Полное количество звёздочек косвенной адресации, равное количеству звёздочек при объявлении указателя, определяет значение переменной.
- Уменьшение количества звёздочек косвенной адресации добавляет к имени переменной слово 'указатель', причём этих слов может быть столько, сколько может быть уровней косвенной адресации для этих имён указателей, то есть столько, сколько звёздочек стоит в объявлении указателя.

Соответствие между количеством уточнений (*) и результатом обращения к значению с помощью указателя

Обращение	Результат обращения	У.К.А.
i	Значение переменной i	0
*ri	Значение переменной, на которую указывает ri	1
ri	Указатель на переменную типа int, значение ri	0
**rri	Значение переменной типа int	2
*rri	Указатель на переменную типа int	1
rri	Указатель на 'указатель на переменную типа int', значение указателя rri	0
***rrri	Значение переменной i типа int	3
**rrri	Указатель на переменную типа int	2
*rrri	Указатель на 'указатель на переменную типа int'	1
rrri	Указатель на 'указатель на 'указатель на переменную типа int'', значение указателя rrri	0

&pppi	address	65000
pppi	data	65001

```
int ***pppi=123=i
int **pppi=65003
int *pppi=65002
int pppi=65001
&pppi=65000
```

i=*pi>**ppi=***pppi=123

&ppi	address	65001
ppi	data	65002

```
int **ppi=i=123
int *ppi=65003
int ppi=65002
&ppi=65001
```

&pi	address	65002
pi	data	65003

```
int *pi=i=123
int pi=65003
&pi=65002
```

&i	address	65003
i	data	123

```
int i=123
&i=65003
```

Операции над указателями

- Присвоить указателю значение адреса данных, или нуль.
- Увеличить (уменьшить) значение указателя
- Прибавить (вычесть) из значения указателя целое число
- Сложить или вычесть значение одного указателя из другого
- Сравнить два указателя с помощью операций отношения.

задать значение Переменной-указателю

- Присвоить указателю адрес переменной, имеющей место в оперативной памяти, или ноль:
- **ptri=&i;**
- **ptri=NULL;**
- Объявить указатель вне функции (в том числе `main`) либо в любой функции, снабдив его описателем **static**, при этом начальным значением указателя является нулевой адрес (**NULL**)

- Присвоить указателю значение другого указателя, который к этому времени уже инициализирован (имеет определённое значение), например: **ptri=ptrj**; -это двойное указание одной и той же переменной.
- Присвоить переменной-указателю значение с помощью функций **malloc** и **calloc**.

Изменение значений указателя

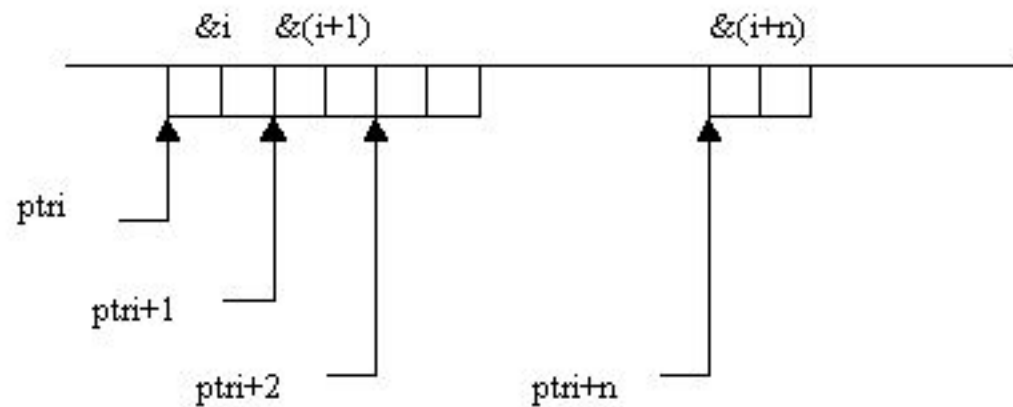
0 +,

0 ++,

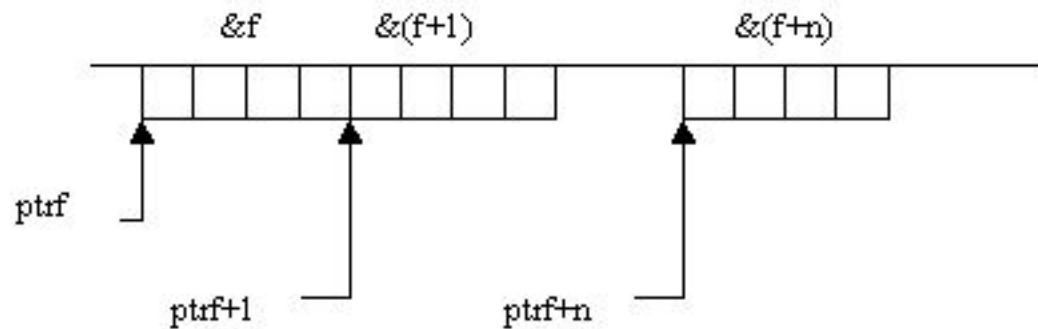
0 -,

0 --

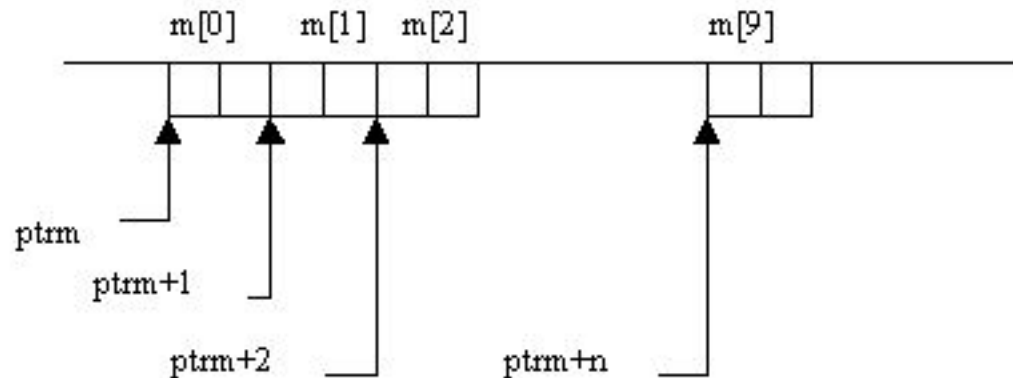

```
int i,*ptri;
ptri=&i;
ptri++;
```



```
float f,*ptrf;
ptrf=&f;
ptrf++;
```



```
int m[10],*ptrm;
ptrm=&m[0];
ptrm++;
```



СВЯЗЬ С МАССИВАМИ

```
0 int mas[10],*ptrm;  
0 ptrm=&mas[0];  
0 *ptrm==mas[0]==*(mas+0) ;  
0 //значение нулевого элемента массива mas  
0 *(ptrm+i)==mas[i]==*(mas+i);  
0 //значение i-го элемента массива mas
```

0 *mas+2==mas[0]+2;

0 *(mas+i)-3==mas[i]-3;

***(&mas[i+1])+2)++;**

- ptrm==&mas[i+1];
- //упрощение выражения, i не играет роли
- ptrm+2==&(mas[i+1])+2;
- //указатель переводится на 2 элемента вперёд
- ***ptrm++==(*ptrm=*ptrm+1);**
- //содержимое ячейки массива извлекается и к нему прибавляется единица

префиксные (слева от имени указателя) постфиксные (справа от имени указателя)

- Префиксные операции в последовательности справа налево.
- Использование значения, полученного после выполнения префиксных операций
- Постфиксная операция над указателем.

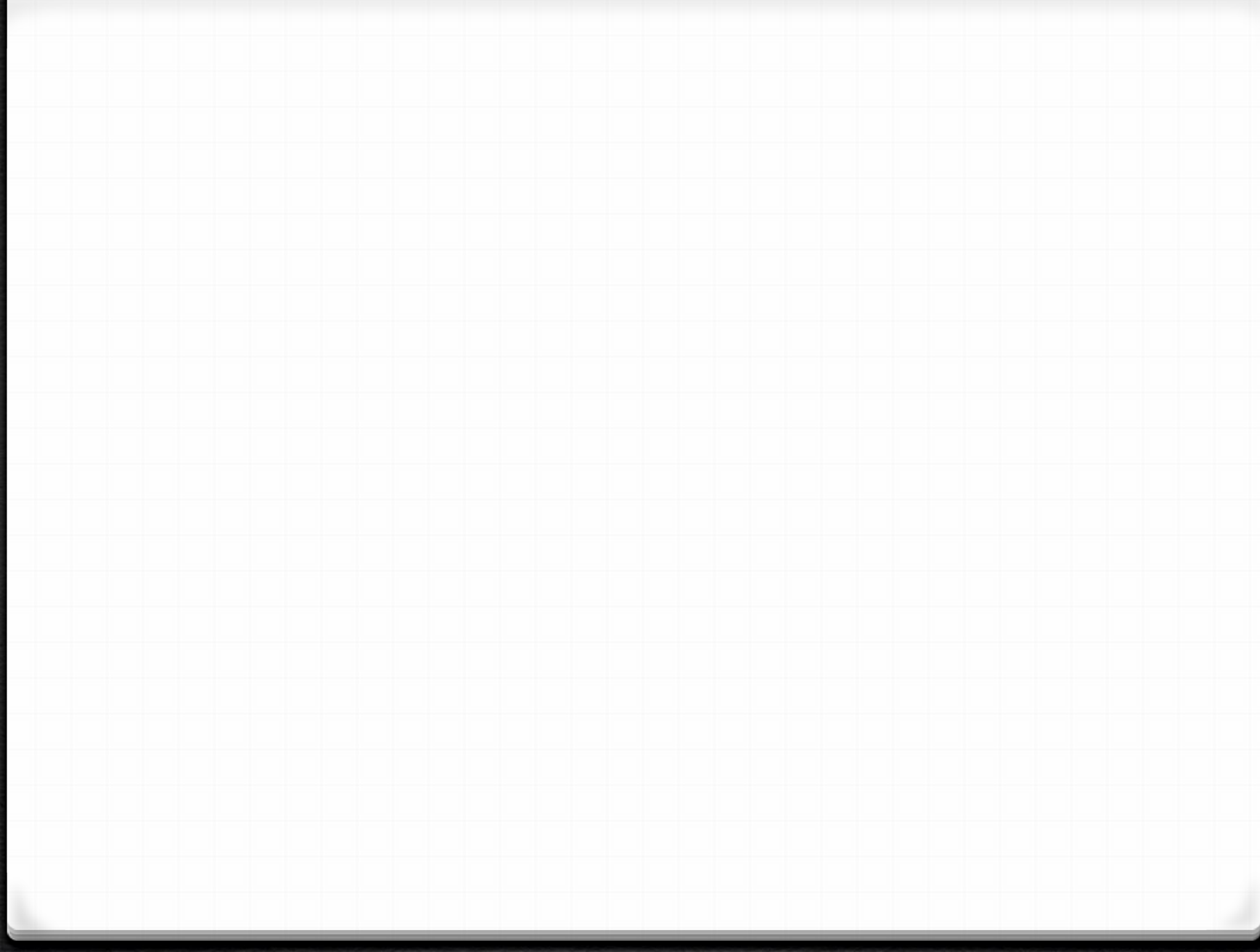
0 ***p++** сначала выполняется префиксная операция над указателем, то есть определяется значение ***p**-содержимое, расположенное по адресу **pх**, а затем выполняется постфиксная операция **++** увеличение значения указателя на квант памяти, то есть на 2 байта (если указатель типа `int`)

0 $(++(*p)+2)$ сначала:

- $*p$ -так как префиксные операции выполняются справа налево.
- $*p=*p+1$ -самая 'левая' префиксная операция
- $+2$ -выполнение постфиксной операции

Проблемы

- Попытка работать с неинициализированными указателям, то есть с указателем, не содержащим адреса оперативной памяти (ОП), выделенной переменной.
- Потеря ссылки, то есть значения указателя из-за присваивания ему нового значения до освобождения ОП, которую он адресует.
- Не освобождение ОП, запрошенной с помощью функции **malloc**



Синтаксис указателей

- 0 `data_type *pointerName;`
- 0 `data_type` — тип данных,
- 0 `pointerName` — имя указателя.

- 0 `int *integerPointer;`

0 // Объявление указателя и простой переменной в одной строке

0 int *pointer1, // это указатель

0 variable; // это обычная переменная типа int

0

0 // Объявление двух указателей в одно строке

0 int *pointer1, // это указатель с именем pointer1

0 *pointer2; // это указатель с именем pointer2

два способа использования указателя

1. Использовать имя указателя без символа *, таким образом можно получить фактический адрес ячейки памяти, куда ссылается указатель.
2. Использовать имя указателя с символом *, это позволит получить значение, хранящееся в памяти. В рамках указателей, у символа * есть техническое название — операция разыменования. По сути, мы принимаем ссылку на какой-то адрес памяти, чтобы получить фактическое значение.

Объявление указателя, получение адреса переменной

- 0 Для того чтобы объявить указатель, который будет ссылаться на переменную,
- 0 необходимо сначала получить адрес этой переменной.
- 0 Чтобы получить адрес памяти переменной (её расположение в памяти), нужно использовать знак & перед именем переменной.
- 0 Это позволяет узнать адрес ячейки памяти, в которой хранится значение переменной.
- 0 Эта операция называется — операция взятия адреса

0 int var = 5;

0 // простое объявление переменной с
предварительной инициализацией

0 int *ptrVar;

0 // объявили указатель, однако он пока ни на что не
указывает

0 ptrVar = &var;

0 // теперь наш указатель ссылается на адрес в памяти,
где хранится число 5

```
0 #include <stdio.h>
0
0 int main()
0 {
0     int var;    // обычная целочисленная переменная
0     int *ptrVar; // целочисленный указатель (ptrVar должен быть типа int, так
    как он будет ссылаться на переменную типа int)
0
0     ptrVar = &var;    // присвоили указателю адрес ячейки в памяти, где
    лежит значение переменной var
0     scanf( "%d", &var ); // в переменную var положили значение, введенное с
    клавиатуры
0     printf( "%d\n", *ptrVar ); // вывод значения через указатель
0     getchar();
0 }
```

0 <http://cppstudio.com/post/9088/>

0 <http://cppstudio.com/post/423/>

0 <http://cppstudio.com/post/429/>